1a)

1. A -> BC giv
2. B -> E giv
3. CD -> EF giv
4. AD -> BCD 1, aug
5. BCD -> BEF 3, aug
6. AD -> BEF 4,5 trans
7. BEF -> F refl
8. AD -> F 6,7 trans

[Alternative]

1) A -> BC given

2) A -> C decomposition, 1

3) AD -> CD augmentation with D, 2

4) CD -> EF given

5) AD -> EF transitivity 3,4

6) AD -> F decomposition 5

bi) Max is m+n by definition of Union.

Min is 0 if m=n=0

(if this is not allowed then min is m if you assume that the relations are identical)

I’d say minimum is max(m,n), this still covers the case where m=n=0

ii) If no common attributes then the natural join becomes a cross join => max is m\*n.

Min is 0 if all attributes are matching but the rows are different

iii) Max is m\*n when all the tuples of R are selected which makes it a normal cross join

Min is 0… when selection returns no tuples, m = 0. The cartesian product of an empty set with anything is empty set.

iv) Max is when one or more attributes satisfy the projection. Therefore you could have m tuples (assuming that no n tuples are the same as m tuplets).

Min is 0 when the projection returns <=m tuples and m<=n

c) (this one seems like a load of bullshit imo)

Union: monotone

Assume R U S. If the tuple is added to either R or S then by the definition of Union the result will contain all the tuples in the previous result as well as the new one.

Intersection: monotone

Assume R ^ S. If the tuple is added to either R or S but doesn’t exist in the other one then there will be no change to the result by the definition of Intersection. On the other hand if the tuple exists in the other relation then the result will have an additional tuple. As the previous result’s tuples will remain intersection is monotone.

Difference: not monotone

Assume R - S. If the tuple is added to S and already exists in R then the result will have one less tuple making it not monotone

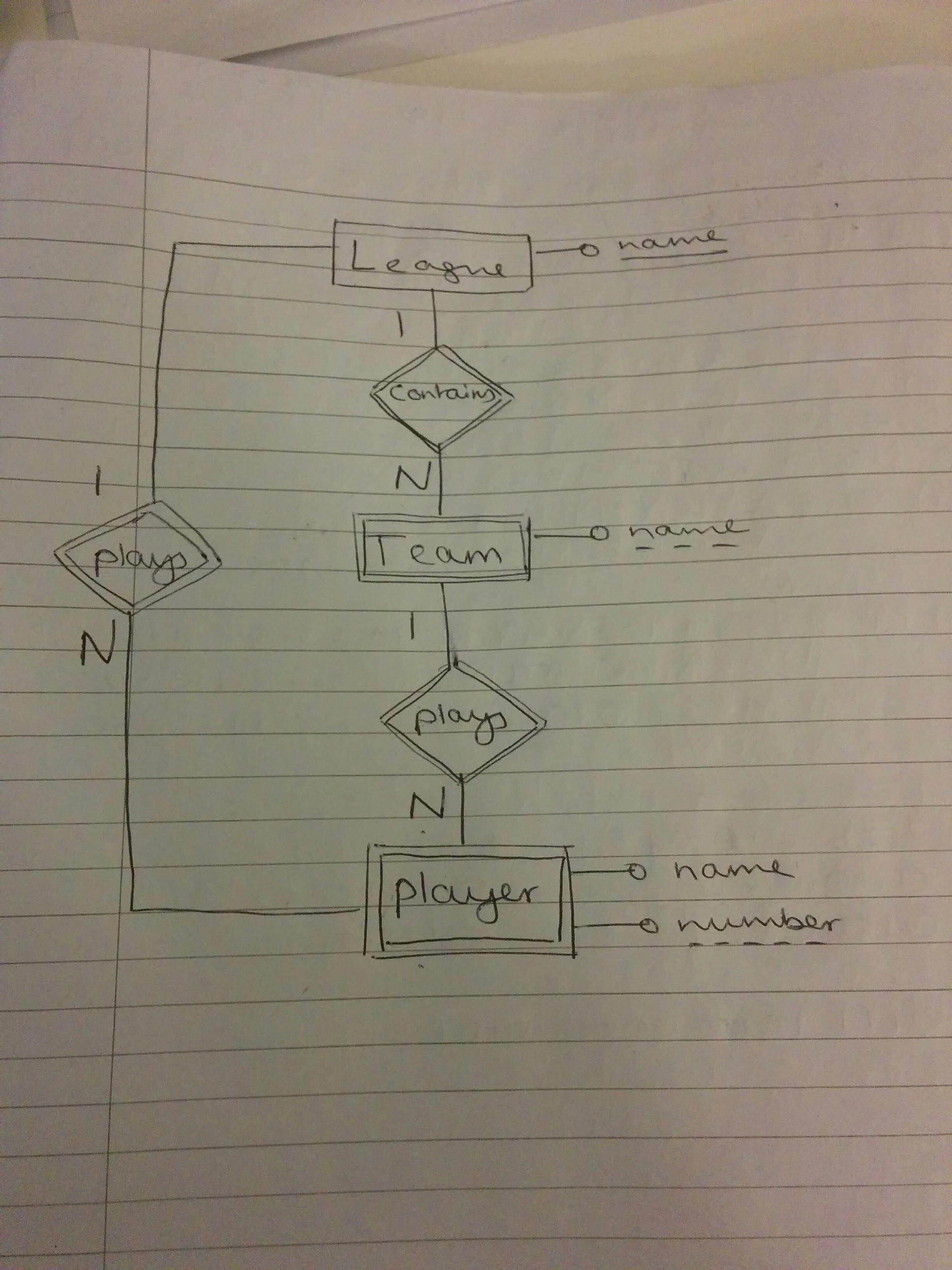
Projection: Monotone

Assume Projection(R). If a tuple is added to R then the new result would be at least the same as the previous result and may also include the new result so is also monotone.

di) A “Weak Entity Set” is an entity set which can’t be uniquely identified by its own attributes alone.

I.e. it’s key relies on attributes from other sets.

A “Strong Entity Set” is an entity set which can be uniquely identified by its own attributes i.e. it has a primary key.



iii) create table league (

name varchar(30)

primary key (name)

)

create table team (

name varchar(30)

derived key (name)

)

create table league\_team (

league\_name varchar(30),

team\_name varchar(30)

)

create table team\_player (

team\_name varchar(30),

player\_number varint(100)

)

create table player (

name varchar(30),

number varint(100)

primary key (name)

)

create table player\_league (

league\_name varchar(30)

)

(Unsure about both of the above tbh. Would appreciate if alternative solutions were posted below)

**Alternative Solution**:

I think this is what a relational schema is as opposed to creating tables in SQL.

league(name)

team(teamName, leagueName)

player(num, teamName, leagueName)

e.g.

create table team(

teamName varchar(120),

leagueName varchar(120),

Primary Key(teamName, leagueName),

Foreign Key (leagueName) references league.name on delete cascade

)

* Do for rest of the weak entity sets.

Worth noting that I think the “plays” relationship set between player and league also avoids a chasm trap (correct me if i’m wrong).

Does anybody know how to use transactions for these sql commands?

2a) Assume that drivers can have more than one car.

SELECT **Count**(DISTINCT driverid)

FROM involved

NATURAL JOIN accident

WHERE **Extract**(year FROM date) = 2011;

b) (Here is my best guess)  
 START TRANSACTION;

SET TRANSACTION SERIALIZABLE;

INSERT INTO accident

VALUES((SELECT (MAX(accidentID) + 1) FROM accident),

‘2012-04-30’,

‘London’)

INSERT INTO involved

VALUES((SELECT MAX(accidentID) FROM accident),

(SELECT carID FROM owns JOIN person USING(driverID) JOIN car USING(carID) WHERE name = ‘Phil Jones’ AND model= ‘Toyota’),

(SELECT driverID FROM person WHERE name = ‘Phil Jones’),

2000)

COMMIT;

c) DELETE

FROM car

WHERE model LIKE ‘%ford%’

AND carid IN

( SELECT carid

FROM owns NATURAL

JOIN person

WHERE name = ‘phil jones’);

(not sure on solution)

Assuming that the relations are linked with all of the ‘ID’s’ attributes as keys

ON DELETE CASCADE

FROM person NATURAL JOIN owns NATURAL JOIN car

WHERE name=’John Smith’ AND model=’Ford’x

This deletion would delete all accidents involving a ‘Ford’ belonging to a ‘John Smith’. There may be multiple John Smith’s in the db and this would erroneously remove the accidents for all of them. Additionally in the case of multi car accidents involving John Smith’s Ford’s, by deleting the accident then the other cars in the accident would have null for their accidentID which would cause errors as it’s the primary key for accident and the foreign key for other relations.

d) SELECT COUNT(\*)

FROM

(SELECT DISTINCT accidentsId

FROM involved NATURAL JOIN person

WHERE name=’Fred Lane’) as fred;

SELECT COUNT (DISTINCT accidentID)

FROM accidents

JOIN involved USING( accidentID)

JOIN person USING (driverID)

WHERE name = ‘Fred lane’

Alternatively:

SELECT COUNT(DISTINCT accidentID)

FROM involved

JOIN person USING (driverID)

WHERE name = “Fred Lane”

e) Assuming accident ‘AC2307’ only involves car ‘Bond007’

UPDATE involved

SET damages= 5000

WHERE carID=’Bond007’

AND accidentID=’AC2307’;